基于 OpenCV 的图像处理技术在验证码文字识别中的应用

宋鹏飞，高文博，张　迪
（中国恩菲工程技术有限公司，北京 100038）

[摘　要] 随着我国有色企业信息化发展的步伐的加快，信息安全是一个不可忽视的重要要素。验证码（CAPTCHA，全称为“Completely Automated Public Turing test to tell Computers and Humans Apart”）是一种用于自动区分计算机和人类的反向图灵测试程序。设计实现一种验证码降噪识别算法模型，对常见类型的验证码进行了识别和分析。研究表明对于字体类型非单一类型的验证码更加难以识别，可用于有色企业信息安全领域。

[关键词] 验证码；信息安全；OpenCV；机器学习
[中图分类号] TP273　　[文献标志码] B　　[文章编号] 1003－8884(2023)06－0054－07
DOI：10.19611/j.cnki.cn11-2919/tg.2023.06.011

1　概述

1.1　相关技术概述及现状

随着有色金属行业的快速发展，我国有色企业亦明显加快了信息化、数字化和智能化发展的步伐。目前，信息技术、数字孪生、人工智能等技术或先进理念在逐步被应用和融合到有色金属产品的设计、生产运行、企业管理、决策支持及市场营销等各个环节中，并带来了显著的效益提升。信息技术已经成为有色金属行业持续发展所不可或缺的要素之一。

在有色行业的信息化实践中，信息安全也是一个不可忽视的重要要素，企业使用的 ERP 和 MES 系统中存储着大量的生产运行数据、排放指标数据、人员组织信息数据等敏感数据。因此需要在登陆环节添加人机验证功能，以防止敏感数据被恶意脚本抓取和外泄。

验证码（CAPTCHA，全称为“Completely Automated Public Turing test to tell Computers and Humans Apart”）是一种用于自动区分计算机和人类的反向图灵测试程序，可以用于防止第三方爬虫程序抓取数据、破解密码或论坛灌水等恶意行为。验证码的生成与测试基于底层数据库和算法无人为干预，在测试中人类可容易通过而计算机几乎不可通过。由于其易于实现和传输负载小等优点，大多数网站在用户登录和注册环节都添加了验证码问答，以验证操作者是否为真实人类。总体来说，在国内外用户数量较稳定的中小型网站中，验证码的应用方式和应用程度基本类似。然而，许多攻击者利用深度神经网络实现了对图像验证码的自动识别，对图像验证码系统的安全性带来了威胁。

OpenCV 是一个在 2000 年由 Apache 基金会发行的开源计算机视觉软件库。其基于 C＋＋语言开发，提供了 Matlab，Ruby，Python 以及 Java 等高级语言的 API 接口，同时支持在 Linux，Windows 等主流 PC 操作系统环境和 ios，Android 等主流手机端操作系统环境上运行，它的轻量级架构和多核处理能力使其实现了基于硬件加速的高效图像处理。OpenCV 被广泛应用在世界各地，拥有超过 4.7 万人的用户社区和超过 1 800 万次的下载记录。

OpenCV 定义的 MAT 多维矩阵广泛地成为了很多项目中图像运算算法在计算机内存中的数据结构。OpenCV 内置了超过 500 种图像处理相关算法，其中包括多种图像滤波函数，几何图像转换函数、绘图函数、颜色模式转换函数、结构分析函数、形

[收稿日期] 2023－09－22
[第一作者] 宋鹏飞（1982—），男，陕西武功人，大学本科，副高级工程师，主要从事管理工作。
[引用格式] 宋鹏飞，高文博，张迪. 基于 OpenCV 的图像处理技术在验证码文字识别中的应用[J]. 有色设备，2023，37(6)：54－60.

状分析函数、对象检测函数、特征检测函数等[1]。

1.2 国内外研究现状

一种基于数字验证码外部轮廓结构特征的数字识别方法[1]针对数字类字符的识别进行了相关研究，将二值化处理后的验证码图像分割成相应若干单个数字图像，然后对分隔出来的单个数字图像外部轮廓 4 个方向进行特征计算，最后依据得到的特征来对字符图像进行识别。一种基于形状上下文理论的验证码识别方法[2]也在验证码识别研究中取得了不错的效果。

1.3 本文研究的验证码图像特点

为了方便使用者识别，有些较简单的验证码由 4～6 个英文字母或数字组成。其他的验证码还有由数学加减法或图片拖拉拽等方式进行反向图灵测试的，不在本文的研究范畴内。本文研究的验证码主要由英文字符和数字字符组成。为了增加机器识别验证码字符的难度，该类验证码对字符进行了旋转，并添加了随机散点噪声和干扰线噪声，还对字符的颜色进行了变换，使得一般的 OCR 算法无法直接对字符进行分类识别。

在验证码的实际使用场景中，由于图片内的文字遭到各种噪声干扰，以及字符本身的随机变形、粘连等，即使人的肉眼也不能保证对每张图片做到一次性准确识别，而是经常需要多次刷新，才能找到方便识别的图片。经过对互联网 20 个主要平台的登陆界面进行调查研究后，笔者认为大多数互联网平台确实支持刷新验证码的操作。因此，对于验证码识别效果，本文将围绕验证码识别所需的主要场景，即登录破解环节，将“全部字符正确识别所需的次数”作为评价标准。

2 基于 OpenCV 的图像预处理研究

2.1 基本架构

本文主要的研究可以分为四个阶段：训练集及测试集的图像样本生成阶段，图像降噪预处理阶段，训练集生成模型阶段和测试集验证模型阶段（图 1）。本文选择生成的验证码样本，统一使用黑体 36 号斜体字体，分辨率为 130 × 40，对象采用的数据结构为 3 通道 MAT 类型，以 PNG 格式进行存储；图像中添加了随机散点噪声，随机干扰线，正弦波型干扰线，还使用了字符旋转和颜色变换，以增加识别的难度。

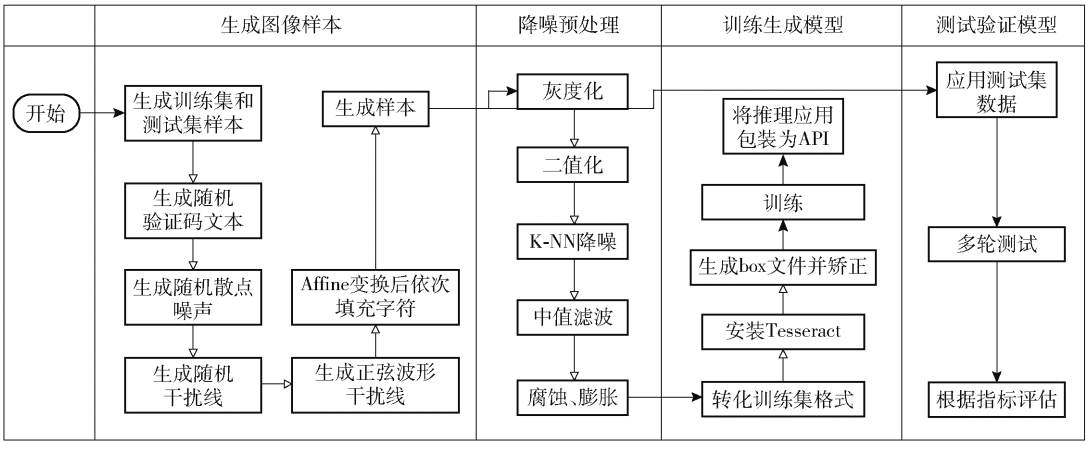

图 1 研究过程示意流程图

使用 Java 语言、JDK1.8 版本实现样本图片生成步骤。

对于随机生成的验证码文本，使用 Random 对象配合当前时间戳，随机产生四个字符，依次填入 StringBuilder 对象中并获取结果字符串。

```
intcodesLen = sources.length();
Random rand = new Random(System.currentTimeMillis());
StringBuilder verifyCode = new StringBuilder(verifySize);
for(int i = 0; i < 4; i++){
    verifyCode.append(sources.charAt(rand.nextInt(codesLen-1)));
}
return verifyCode.toString();
```

对于随机干扰线，将随机线条的颜色限制在

[160,200]区间范围内。

```
//绘制干扰线
    Graphics2D g2 = image.createGraphics();
Random random = new Random();//coded by gaowb;
g2.setColor(getRandColor(160, 200));// 设置线条的颜色
for (int i = 0; i < 20; i++) {
    int x = random.nextInt(w-1);
    int y = random.nextInt(h-1);
    int xl = random.nextInt(6) + 1;
    int yl = random.nextInt(12) + 1;
    g2.drawLine(x, y, x + xl + 30, y + yl + 20);
}
```

对于随机噪点,首先设置一个固定的噪声率,以浮点数方式声明;噪声率乘以图片的长宽后,采用 getRandom Color()方法随机得到一个噪点颜色,然后使用 image.setRGB()方法在图像的长宽范围内添加像素点。

```
//添加噪点
float noiseRate = 0.07f;// 噪声率
int area = (int) (noiseRate * w * h);
for (int i = 0; i < area; i++) {
    int x = random.nextInt(w);
    int y = random.nextInt(h);
    int rgb = getRandomIntColor();
    image.setRGB(x, y, rgb);
}
```

对于正弦波型干扰线,使用 Math.sin()方法,配合两个随机浮点数变量,生成正弦波函数对应的波形干扰线所覆盖的对应区域,并使用 Graphics2D.drawLine()方法,在对应坐标区间上描绘深色干扰线。

```
intperiod = random.nextInt(2);
boolean borderGap = true;
int frames = 1;
int phase = random.nextInt(2);
for (int i = 0; i < h1; i++) {
    double d = (double) (period >> 1)
            * Math.sin((double) i / (double) period
            + (6.2831853071795862D * (double) phase)
            / (double) frames); //coded by gaowb;
    g.copyArea(0, i, w1, 1, (int) d, 0);
    if (borderGap) {
        g.setColor(c);
        g.drawLine((int) d, i, 0, i);
        g.drawLine((int) d + w1, i, w1, i);
    }
}
```

对于随机出来的验证码文本,将其设置为黑体字体,字体大小稍小于图片高度,并对每个字符使用 AffineTransform.setRotation()方法和 Graphics2D.setTransform()进行 affine 变换,以实现对字符的旋转效果。

```
intfontSize = h-4;
Font font = new Font("黑体", Font.ITALIC, fontSize);
g2.setFont(font);//coded by gaowb;
char[] chars = code.toCharArray();
for(int i = 0; i < verifySize; i++){
    AffineTransform affine = new AffineTransform();
    affine.setToRotation(Math.PI / 4 * rand.nextDouble() * (rand.nextBoolean() ? 1 : -1), (w / verifySize) * i + fontSize/2, h/2);
    g2.setTransform(affine);
    g2.drawChars(chars, i, 1, ((w-10) / verifySize) * i + 5, h/2 + fontSize/2-10);
}
```

总体上,属于文字本身的有效信息区域,其颜色比较深,像素点的颜色属于[100,160]的范围;而背景色相对较浅,像素点的颜色属于[200, 250]的范围内。每个字符虽仍按照应有的顺序从左至右依次排列,但字符间经 affine 变换后经常出现互相覆盖重叠的现象,见图 2。

2.2 图像预处理

在图像降噪预处理阶段,本文模拟图像识别的一般步骤,通过多种算法剔除或降低图像中的各种干扰信息。这些干扰信息处理算法本质上是各种滤波的叠加,将图像归一化,以便于模型的训练。对于

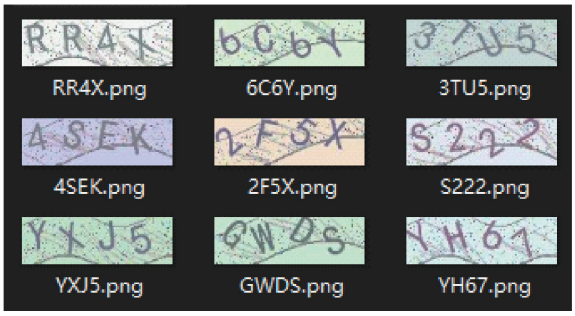

图 2　验证码示例

这个过程，本文使用 Python3 作为实现用编程语言。

本文通过 OpenCV 的内置 cvtColor( ) 函数将图像中的所有像素点都转化为范围属于[0,255]的灰度值。其中 0 为纯黑色，255 为纯白色，颜色根据数值依次变浅。

```
#灰度化
gray_img = cv2.cvtColor(origin_img, cv2.COLOR_BGR2GRAY)
cv2.imshow('gray', gray_img)
```

这样处理后，所有的像素点所携带的信息都只剩下了颜色深浅的区别，如图 3 所示。

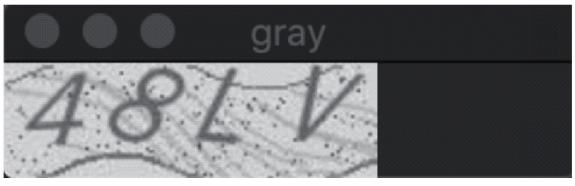

图 3　灰度化处理后的验证码图像示例

之后将转化后的灰度值经过某一阈值过滤，以对灰度化图像进行二值化处理。经过多次实验，本文选择 170 作为合适的阈值参数值，将所有的像素点转化为 0 或 255 的极值，这样所有数值高于 170 的像素点都会变为纯白色，而低于 170 的像素点变为纯黑。

```
#二值化
ret, bin_img = cv2.threshold(gray_img, 170, 255, cv2.THRESH_BINARY);
cv2.imshow('binary', bin_img)
```

经过二值化处理后的图像，可以去除大部分的浅色干扰线噪声，如图 4 所示。

对于剩下的随机散点噪声，本文选择使用 k-邻域算法(K-NN)来处理这些颗粒度较小的像素点集合，以实现进一步降噪。

图 4　二值化处理后的验证码图像示例

K-NN 算法用作分类算法的核心思想是：在给定的集合 T 中，找到与实例的特征向量 x 最临近的 n 个点，根据决策规则对筛选出符合的临近点，并对实例进行分类。

在本文的案例中，假设每个卷积核的边长为 3，那么，除了图片的四条边界上的像素点外，在每个其余的像素点的邻域中都可以找到 8 个距离该像素点的切比雪夫距离(Chebyshev Distance)为 1 的“临近点”。只要这 8 个“临近点”中的黑色像素点，即数值为 0 的像素点数量，大于等于 k 个，就认为该像素点是属于文字部分的有效像素点，因为与它相连的黑色点数量较多；如果邻近的黑色像素点数量小于 k 个，就认为它是噪声像素点，把它的数值设置为 255，将其变为白色。算法实现的核心代码如下所示。

```
#获取图像的宽,高数值
w, h = bin_img.shape
    for w1 in range(w):
        for h1 in range(h):
            # 如果是图像边界区域,不做计算
            if w1 == 0 or h1 == 0:
                bin_img[w1, h1] = 255
                continue
            #计算邻域中像素值小于 255 的个数
            pixel = bin_img[w1, h1]
            if pixel == 255:
                continue
            #调用 K 邻域降噪算法
            #如果临近点 0 数量小于 k,清除
             if noise_pixels_count(bin_img, w1, h1) < k:
                bin_img[w1, h1] = 255
    # coded by gwb
    # KNN 降噪
    def noise_pixels_count(img_obj, w, h):
```

```
count = 0
width, height = img_obj.shape
for w1 in [w-1, w, w+1]:
    for h1 in [h-1, h, h+1]:
        if w1 > width -1:
            continue
        if h1 > height - 1:
            continue
        if w1 == w and h1 == h:
            continue
        #二值化的像素值设置为 255
        if img_obj[w1, h1] < 255:
            count += 1
return count
```

经过多次实验研究,发现如果 k 值调的过高,会将一些比较细长的有效字符,如‘1’或‘I’等,错误地消除;如果 k 值过低,则无法去除足够的噪声。最终,本文参数 k 的值定为 4,得到了不错的降噪效果,如图 5 所示。

图 5　4-邻域降噪算法函数处理后的验证码图像示例

降噪处理进行到这一步,距离可用的降噪结果已经很接近了,而剩下的条形残留噪声仍旧会导致一些错误的识别。经过研究,本文选择使用中值滤波算法,配合 3×3 的卷积核尺寸进行进一步降噪。在这之后,又使用了 OpenCV 内置的腐蚀函数和膨胀函数,进一步削弱残余的噪声。经研究,这两个函数配合 2×2 的卷积核可以有效地进一步削弱噪声,同时不影响有效的文字部分,帮助 OCR 识别模型可以以较高的识别率识别目标文字。

```
#中值滤波,卷积核尺寸 3*3
img = cv2.medianBlur(bin_img, 3)
#先使用腐蚀,卷积核尺寸 2*2
im1 = cv2.dilate(img, np.ones((2, 2)))
#再使用膨胀,卷积核尺寸 2*2
im2 = cv2.erode(im1, np.ones((2, 2)))
```

进一步降噪处理后的效果如图 5,可以看到的显著区别在示例图片图 6 的右下角,残留的干扰线更加难以别错误地识别为字符“i”。

图 6　中值滤波函数、腐蚀函数、膨胀函数处理后的验证码图像示例

3　基于 Tesseract 的识别算法训练及推理效果

Tesseract 是一个开源 OCR 文字识别引擎,最早由 HP 实验室发布,其源码在 Github 上开源,目前由 Google 持续维护。Tesseract 主要针对印刷字体的 OCR 识别,支持多种语言,文本库拥有庞大的 Unicode 字符识别积累,同时支持在多平台移植。Tesseract 采用了一系列图像处理、特征提取和机器学习技术来实现文字识别过程,其中特征提取主要是依靠对卷积神经网络(CNN)的训练,而第四代的 Tesseract 添加了长短期记忆网络(Long Short Term Memory)来对文本图像进行行识别和字符分割等操作[4],进一步地提高了 Tesseract 的识别效果和识别率。目前,该引擎已经支持超过 110 种语言字符。

针对训练集,本文使用第二章的验证码样本生成方式,构造了 1 000 张训练集样本图片,以. png 格式进行存储。为了方便训练,本文调用图像预处理相关算法,将训练样本集进行统一灰度化、二值化处理,并进行降噪,去除干扰线。为了方便后续处理,将所有预处理后的图片以. tiff 格式存储。

JTessBoxEditor 是一款专用于 Tesseract 的训练矫正工具软件。它可以用于训练处理后图片,来生成定制的识别字体库。本文使用该软件的 Merge Tiff 功能将所有. tiff 文件合并为. tif 文件保存。

在安装好 Tesseract 和相关依赖如 LibTiff 和 Leptonica 后,使用 Tesseract 的 makebox 指令对样本生成初始化生成. box 文件,该文件是 Tesseract 初步的行识别结果,里面存储了每个字符在每张对应图片中矩形框的坐标及大小等信息。

由于图像中的字符并非简单黑体字符,而是经过 affine 变换旋转的特殊字符,因此需要对旋转后的字符进行标记,使本模型的字体库更加充分。在该文件夹下继续使用 JTessBoxEditor 软件,可以可视

化地对每个矩形框坐标进行调整纠正并手动标记。

最后，本文将该. box 文件和预处理的样本训练集. tif 文件使用 Tesseract 的 train 功能进行训练，并导出模型文件，作为该系统下 Tesseract 的默认 OCR 执行模型。

在 Python 环境中引入 pytesseract 库、PIL 库和 flask 库后，将图像预处理过程和模型识别过程包装为 API 接口，方便 OCR 识别测试。在测试过程中，该脚本被作为进程启动后台挂起，每当接受对一个接口请求时，该进程将 request 上传的. png 图片进行解析并缓存在文件系统中，再通过 pytesseract 库调用本地的 tesseract 服务并执行 OCR 识别，将结果字符串以 response 方式返回。核心代码如下所示。

```
import importlib,sys)
import os
from flask import request, Flask
from PIL import Image
import pytesseract

app = Flask(__name__)

@app.route("/yanzhengma", methods=['POST'])
def get_frame():
    uploaded = request.files["file"]
    path = os.path.abspath("./")
    app.config['tmp'] = path
    # coded by gaowb
    uploaded.save(os.path.join(app.config['tmp'],
'tmp.png'))
    if uploaded:
        result = code_breaker(path + "/tmp.
png", 4)
        return str(result)
    else:
        return "-1"

def code_breaker(image):
    result = pytesseract.image_to_string(image,
lang="eng", config='--psm 10')
        print('result:' + result)
    return result

if __name__ == "__main__":
    app.run(IP, PORT)
```

针对测试集，本文使用和第二章相同的方式随机生成了 4 000 张验证码 png 图片作为测试集，将其发送至 OCR 识别接口进行测试。在对测试集经过分批多轮测试后，本文发现，“全部字符正确识别所需的次数”指标的中位数和平均数稳定在 2 次。该模型对旋转后的字符和不端正的字符具有较良好的识别能力，能够排除绝大部分噪声带来的干扰；对于部分旋转后出界的字符，和与其他字符或干扰线过分重叠的字符，准确识别的识别率较低。

4 结束语

有色金属行业是国民经济发展地重要组成部分之一，许多现代工业领域，如航空、航天和半导体行业都依赖有色金属行业的原材料供应。因此，加强该行业对信息管理系统和相关控制系统的信息安全管理就显得尤为重要。本文研究的目的旨在为有色行业的验证登录环节提供优化建议。

在生成验证码样本的过程中，笔者使用了单一的字体类型。在对字体替换后的不同样本进行测试时，发现不同的字体也会对本文训练的模型带来额外的误差率。

针对如何提高 OCR 识别率的问题，下一步的研究工作是考虑如何通过更好的归一化方式消除不同字体带来的误差，或者如何高效的提升模型对多种字体的适应能力。

针对在有色行业企业信息系统中，如何在登陆验证环节中提升人机识别的反向图灵测试效果的问题，基于以上研究，我们可以在验证码生成的方法中添加更多的字体种类，将不同字体随机混合在一起呈现，同时让干扰线更多地和字符重叠，比如提高正弦波形干扰线的变化频率等方式，以尽最大可能干扰验证码字符被 OCR 算法识别的效果。

[参考文献]

[1] 潘大夫，汪渤. 一种基于外部轮廓的数字验证码识别方法[J]. 微计算机信息，2007(25)：256－258.

[2] 苏磊，马良. 形状上下文在验证码识别中的应用[J]. 微计算机信息，2007(35)：252－253＋246.

[3] 王璐. 验证码识别技术研究[D]. 合肥：中国科学技术大学，2023.

[4] Google. Tesseract-OCR. tessdoc[EB/OL]. (2023－11－29)[2023－11－29]. https://tesseract-ocr. github. io/

tessdoc/Installation. html.
[5]OpenCV. OpenCV[EB/OL]. (2023 - 11 - 29)[2023 - 11 - 29]. https://opencv. org/.
[6] 张晓菲,邹婷,刘培鹤,等. 基于字符型验证码破解算法研究[J]. 北京电子科技学院学报,2016,24(4):50 - 54 + 77.
[7] 杨晔逵. 我国有色金属企业信息化建设研究[J]. 湖南有色金属,2010,26(1):75 - 78.

OpenCV-based image processing technology: application in text recognition of CAPTCHA

SONG Pengfei, GAO Wenbo, ZHANG Di

Abstract: As the non-ferrous enterprises in China accelerate their development of information technology, information security is an important factor that cannot be ignored. CAPTCHA is a reversed Turing test program which could be used to autonomously distinguish between computers and human. A captcha noise reduction recognition algorithm model was designed and implemented, and a common type of CAPTCHAs were identified and analyzed. Studies have shown that for CAPTCHAs with multiple font types are more difficult to identify, thus could be used to improve information security in the non-ferrous industry.
Key words: CAPTCHA; information security; OpenCV; machine learning

(上接第 53 页)

The application of multicast and unicast in industrial video surveillance systems

YIN Wancong

Abstract: This article explores the applications of multicast and unicast in industrial video surveillance systems. This article analyzes unicast multicast applications based on different network topologies and verifies them through experiments. The main conclusion of this study is that as the number of network nodes continues to increase, multicast algorithms perform better than unicast algorithms, and multicast algorithms can achieve lower network bandwidth utilization.
Key words: industrial video surveillance system; multicast; unicast; network topology structure; transmission optimization